Handle User Inputs and Forms

Making Requests from HTML Pages to PHP APIs (Forms)

- Example: Communication through a form and POST method
    - Backend Code: submit.php
    - Frontend Code: test2.html

Example: Communication through a form and POST method

Frontend (`test2.html`) $\longleftrightarrow$ **Backend** (`submit.php`)

- In this section, we use a form and POST to request JavaScript.
- PHP server handles the POST request to return a JSON string as a response.

Backend Code: submit.php

<?php switch ($method) { case 'POST': // Handle POST parameters $name = $_POST['name'] ?? ''; $email = $_POST['email'] ?? ''; $info = ['name' => $name, 'email' => $email]; sendResponse($info, 'Response from POST request'); break; Default: sendError('Method Not Supported', 404); break; } ?>

Frontend Code: test2.html

- HTML form

```
<form action="submit.php" method="post">
  Name: <input type="text" name="name"><br>
  Email: <input type="email" name="email"><br>
  <input type="submit" value="Submit">
</form>
```

JavaScript

```javascript
document.querySelector('form').addEventListener('submit', async function (event) {
   event.preventDefault(); // Prevent the default form submission
   // 1. make a request
   // 2. get the response
   // 3. display results
});
```

- We use the listener to process the form in JavaScript.

Why Use a JavaScript Listener for Form Submission?

A listener on the `'submit'` event allows you to control what happens when a form is submitted fully.

1. Preventing Default Behavior

- By default, submitting a form reloads the page.
- A listener uses `event.preventDefault()` to stop the reload.
- ✅ This enables smoother interaction without leaving the page.

2. Handling Data with JavaScript

- The listener lets you capture form input using the `FormData` API.
- You can:
  - Validate input
  - Modify or append data
  - Prepare it before sending
- ✅ More control over what is sent to the server.

3. Asynchronous Submission (AJAX)

- With a listener, you can submit data via `fetch()` or `XMLHttpRequest`.
- This avoids page reloads.
- ✅ Makes the app feel faster and more dynamic.

4. Custom Logic and User Feedback

- You can:
  - Show loading indicators
  - Display success or error messages
  - Update the UI based on the server response
- ✅ Great for real-time feedback and polished user experience.

1. Make a request

- The `event.target` has the form for making the request.

```javascript
const form = event.target;
const formData = new FormData(form);
const response = await fetch(form.action, {
  method: 'POST',
  body: formData
});
```

2. get the response

```
const data = await response.json();
```

3. display results

```javascript
try {
  document.getElementById('result').textContent =
    JSON.stringify(data, null, 2);
} catch (error) {
  document.getElementById('result').textContent =
    'Error: ' + error.message;
}
```

Complete JavaScript code

```
    document.querySelector('form').addEventListener('submit', async function (event) {
      event.preventDefault(); // Prevent the default form submission

      const form = event.target;
      const formData = new FormData(form);

      try {
        const response = await fetch(form.action, {
          method: 'POST',
          body: formData
        });

        const data = await response.json();

        // Display the result
        document.getElementById('result').textContent =
          JSON.stringify(data, null, 2);
      } catch (error) {
        document.getElementById('result').textContent =
          'Error: ' + error.message;
      }
    });
  </script>
```